**Google Protocol Buffer**

1. **What are “Protocol Buffers”?**

Protocol buffers are a flexible, efficient, automated mechanism for serializing structured data – think XML, but smaller, faster, and simpler. You define how you want your data to be structured once, then you can use special generated source code to easily write and read your structured data to and from a variety of data streams and using a variety of languages. You can even update your data structure without breaking deployed programs that are compiled against the "old" format.

1. **Why “Protocol Buffer”?**

Protocol buffers have many advantages over XML for serializing structured data. Protocol buffers:

* are simpler
* are 3 to 10 times smaller
* are 20 to 100 times faster
* are less ambiguous
* generate data access classes that are easier to use programmatically

For example, let's say you want to model a person with a name and an email. In XML, you need to do:

*<person>*

*<name>John Doe</name>*

*<email>jdoe@example.com</email>*

*</person>*

When this message is encoded to the protocol buffer binary format it would probably be 28 bytes long and take around 100-200 nanoseconds to parse. The XML version is at least 69 bytes if you remove whitespace, and would take around 5,000-10,000 nanoseconds to parse.

1. **Maked .proto file**

* **Define a message type:**

*message RequestMessage*

*{*

*required string content = 1;*

*required string perReceipt = 2;*

*option string perSend = 3;*

*}*

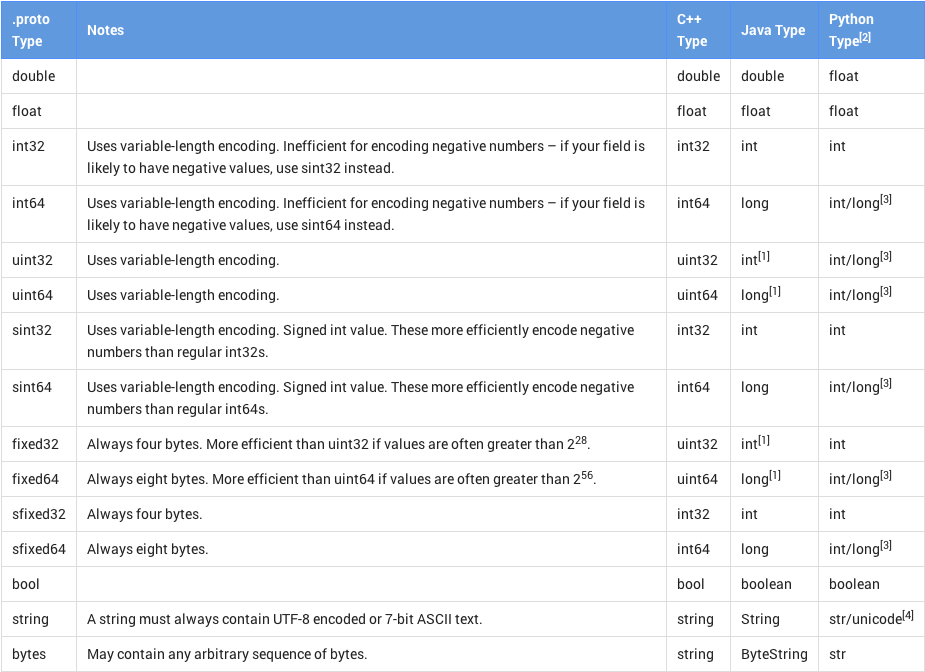
* **Assigning tags**

Each field in the message definition has a unique numbered tag. These tags are used to identify your fields in the message binary format

* **Specifying fields rules**

You specify that message fields are one of the following:

* *required*: a well-formed message must have exactly one of this field.
* *optional*: a well-formed message can have zero or one of this field (but not more than one).
* *repeated*: this field can be repeated any number of times (including zero) in a well-formed message. The order of the repeated values will be preserved.
* **Value types**



* **Option fields and default value**

*optional int32 result\_per\_page = 3 [default = 10];*

* **Enumeration**

*enum MessageType*

*{*

*MESSAGE\_REQUEST = 1;*

*MESSAGE\_REPONSE = 2;*

*MESSAGE\_USES = 3;*

*}*

* **Related Messages**

Using as field type, when you want to include other message type to your message:

*message SearchResponse {*

*repeated Result result = 1;*

*}*

*message Result {*

*required string url = 1;*

*optional string title = 2;*

*repeated string snippets = 3;*

*}*

* **Importing definitions**

*import "myproject/other\_protos.proto";*

* **Extensions**

Extensions let you declare that a range of field numbers in a message are available for third-party extensions. Other people can declare new fields for your message type with those numeric tags in their own .proto files without having to edit the original file. Example:

*message Foo {*

*// ...*

*extensions 100 to 199;*

*}*

This says that the range of field numbers [100, 199] in Foo is reserved for extensions. Other users can now add new fields to Foo in their own .proto files that import your .proto, using tags within your specified range – for example:

*extend Foo {*

*optional int32 bar = 126;*

*}*

* **Genarating Classes in Obj-C**

*protoc --proto\_path=IMPORT\_PATH --cpp\_out=DST\_DIR --java\_out=DST\_DIR --python\_out=DST\_DIR path/to/file.proto*

* IMPORT\_PATH specifies a directory in which to look for .proto files when resolving import directives. If omitted, the current directory is used. Multiple import directories can be specified by passing the --proto\_path option multiple times; they will be searched in order. -I=IMPORT\_PATH can be used as a short form of --proto\_path.
* You can provide one or more output directives:
  + --cpp\_out generates C++ code in DST\_DIR. See the C++ generated code reference for more.
  + --java\_out generates Java code in DST\_DIR. See the Java generated code reference for more.
  + --python\_out generates Python code in DST\_DIR. See the Python generated code reference for more.
* As an extra convenience, if the DST\_DIR ends in .zip or .jar, the compiler will write the output to a single ZIP-format archive file with the given name. .jar outputs will also be given a manifest file as required by the Java JAR specification. Note that if the output archive already exists, it will be overwritten; the compiler is not smart enough to add files to an existing archive.
* You must provide one or more .proto files as input. Multiple .proto files can be specified at once. Although the files are named relative to the current directory, each file must reside in one of the IMPORT\_PATHs so that the compiler can determine its canonical name.